02

# Lógica

**Transcrição**

[00:00] Chegou a hora de implementarmos a lógica de movimentação dos fantasmas. Vamos fazer isso implementando uma primeira inteligência para o nosso fantasma, que eu nem sei se chamaria muito de inteligência, porque primeiro ele vai sempre andar para a direita. Fazendo esse movimento, depois podemos criar coisas mais complexas.

[00:35] A primeira coisa é que vamos usar o mapa dois, porque ele tem mais graça. Quando jogamos, a cada nova rodada, depois de movimentar nosso herói, vamos mover fantasmas no nosso mapa. Mover fantasmas é uma lógica de negócios. O caractere do fantasma é a letra F. O que eu tenho que fazer para movimentar todos eles? Primeiro tenho que encontrar. Fazemos isso usando o each_with_index. Ele recebe a linha atual e a linha, que a é o número da minha linha.

[02:11] Antes usávamos a posição do caractere do herói, o index, para saber se ele estava lá ou não. Agora isso não faz sentido, porque se eu tentar acessar um único fantasma nessa linha, só vou achar um, e logo de cara sabemos que temos dois fantasmas. Agora sim vou precisar varrer a linha inteira, procurando cada um deles, e a coluna.

[02:50] Se é um fantasma, vamos movê-lo. Eu sei disso se o caractere atual for igual ao do fantasma. Para mover, vamos precisar de um mapa. Vou pegar a linha e a coluna em que ele está e vou colocar um espaço em branco. Vou pegar a linha mais igual a zero, coluna mais igual a um, mapa nessa linha e coluna agora é um fantasma. Uma maneira bem simples de mover para a direita.

[04:12] Deu erro. Ele diz que a função each_with_index não existe. Nós usamos quando acessamos o mapa, mas ele deu o erro na linha 59, porque a string não tem um método. Ela só tem o each. Mas array e string não é a mesma coisa? Por trás dos panos, pode até ser uma array de caracteres, porém string não é array. String tem diversos comportamentos, métodos, funções que uma array tem, mas não é a mesma coisa.

[05:35] Antes, eu precisava de um cara que tivesse um método size, colchetes, e o each_with_index. Cada uma das coisas que estavam dentro desse mapa também precisavam do colchetes, do size, e só. Só que agora preciso que cada cara de dentro tenha também o each_with_index. E aí minha aplicação quebrou. Você lembra que eu falei que se ele fala que nem um pato, anda que nem um pato, nada que nem um pato, para mim ele é um pato? Não era bem verdade. É verdade de um ponto de vista, ele é alguém que tem o each, que tem o size, o colchete, a string tem essas coisas, mas no momento em que cobrei um pouco mais dele, achando que ele fosse um array, ele não é.

[06:47] Vemos aqui na prática como o duck typing funciona. Até o momento não me preocupei que ele fosse realmente uma array. Só precisava de determinados métodos que por acaso um outro cara de outro tipo também tem, e funciona da mesma maneira. A semântica desses métodos, desses comportamentos era a mesma. No momento em que eu precisei de um comportamento a mais, depende.

[07:35] Na prática, por mais que eu só esteja interessado se ele tem ou não esse comportamento, o interpretador só está interessado se existe ou não. Para o interpretador, o duck typing é o suficiente. Mas isso só é importante para o interpretador. Para nós, muito mais é importante. Mais do que saber que ele tem a função, preciso ter certeza que o each_with_index que ele tem faz exatamente o que quero.

[08:12] Para saber que ele faz o que quero, preciso saber quem é implementação do each_with_index. Para isso, preciso saber o tipo da minha linha atual. Eu como desenvolvedor tenho que saber o tipo da minha variável. Para o interpretador não. Quando falamos duck typing, por mais que estejamos digitando de uma maneira, quem está sendo duck é o interpretador. Eu como programador estou totalmente interessado no tipo, porque o tipo diz quais

comportamentos existem e o tipo diz o que esses comportamentos fazem, que é fundamental. Não adianta chamar a função e fazer uma coisa totalmente diferente.

[08:52] Eu como desenvolvedor estou interessado nos tipos, tenho que saber que tipo está indo para um lado e para o outro, porque senão não tem o comportamento que vou precisar, senão o comportamento que ele vai executar em tempo de execução é diferente daquele que eu tinha imaginado. Eu como desenvolvedor estou interessado nos tipos, o interpretador não está interessado. Ele só quer saber se o comportamento existe com aquele nome.

[09:30] E aí para nós o que acontece? Até agora funcionou perfeito e você pode argumentar que agora é o momento de mudar. Em outras linguagens, onde tivéssemos que nos preocupar com a tipagem para o compilador ou interpretador antes, teríamos que ter nos preocupado antes, o que pode ser visto como vantagem ou desvantagem.

[09:57] Agora vamos ter que pensar que a linha atual tem que ser uma array de caracteres. Como transformar uma string em uma array de caracteres? Tem um método chamado chars que devolve uma array de caracteres. Se é uma array de caracteres, ele tem o each_with_index. Percebe como a tipagem é extremamente importante para o desenvolvedor? Se você olhar esse código, tudo está ligado a depender que o mapa seja uma array de strings, senão não funciona. Seria muita coincidência que no duck typing eu tivesse um cara que se comporta como pato, anda como pato, nada como pato, grasna como pato, faz tudo que nem pato, mas que não é pato.

[11:11] O desenvolvedor depende do tipo. O interpretador não está nem aí. Isso é importante de lembrar. E existem maneiras de garantir isso com testes. O que eu queria focar é que o duck typing mostrou que precisamos de alguém com each_with_index, estávamos atrelados a um tipo determinado, mas na verdade precisávamos de outro tipo. E aí a aplicação gritou. Eu poderia ter pego isso com teste automatizado ou em produção. Talvez nem o teste pegaria.

[12:20] Os dois fantasmas agora movem para a direita. Está funcionando. Mas o fantasma vai para cima do muro. Faltou validar o movimento.